EUROPEAN OFFICE OF AEROSPACE RESEARCH AND DEVELOPMENT

Contract F61775-00-WE022

23 September, 2000

FINAL REPORT,

17 April, 2001

NEURAL NETWORKS FOR ROBOT CONTROL

Principal Investigator: Prof. Chaïban NASR

Lebanese University - Faculty of Engineering, Section I
Dept. of Electrical and Electronic Engineering
P.O. Box N^{o} 7 - Zgharta - North Lebanon
LEBANON

Phone: (961) 3 369245
Fax: (961) 6 385089
E-mail: chnasr@ul.edu.lb

1-INTRODUCTION:

Robotic manipulators are highly non-linear, dynamically coupled, multi-axis electromechanical systems that are commonly used in tasks such as welding, material handling in manufacturing, and as accurate positioning systems. In these tasks, the end-effectors are required to move from one place to another, or to follow some desired trajectories as close as possible at a fast operational speed in a free workspace. Trajectory tracking control of robots is, thus, of practical significance, and is the simplest yet the most fundamental task in robot control [1]. Concurrent advances in microprocessor technology have made the implementation of complicated non-linear control algorithms practically feasible. Robots may have to manipulate loads of different weight, size and mass distributions, in which case, the dynamics of the robots will also be different. Controllers designed for a particular nominal payload may not be able to control the system properly for all changes in the parameters. Some model based adaptive controllers have been proposed based on linear in the parameters models. The controllers are both stable and robust under certain conditions. The system can achieve asymptotic tracking without using infinite-gain feedback through on-line parameter adaptation when the system is subject to parametric uncertainties only [2]. However, instability may occur due to

Form SF298 Citation Data

Report Date *("DD MON YYYY")* 20042001	**Report Type** Final	**Dates Covered (from... to)** *("DD MON YYYY")*

Title and Subtitle Neural Networks For Robot Control	**Contract or Grant Number** F61775-00-WE022
	Program Element Number
Authors Nasr, Chaiban	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Lebanese University-Faculty of Engineering, Section I El-Kobbeh Tripoli North Lebanon	**Performing Organization Number(s)**
Sponsoring/Monitoring Agency Name(s) and Address(es) EOARD PSC 802 BOX 14 FPO 09499-0200	**Monitoring Agency Acronym**
	Monitoring Agency Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	
Abstract This report results from a contract tasking Lebanese University-Faculty of Engineering, Section I as follows: The proposed research consists of the following: a) Application of artificial neural networks (multi-layer perceptrons, MLPs) for 2D planar robot arm by using the dynamic backpropagation methods for the adjustment of parameters; and optimization of the architecture; b) Application of artificial neural networks in controlling closed-loop 2D planar robot arm and comparison with the use of proportional-integral-differential (PID) controllers. The results should create a basis for a further research program connecting the fundamental knowledge with practical applications that can subsequently include: 1) Theoretical and experimental studies in controlling dynamic robot arms by using neural networks in real-time process; 2) Research of optimal architectures used in closed-loop systems in order to compare with adaptive and robust control.	
Subject Terms EOARD; Control; Robotics; Neural Networks	
Document Classification unclassified	**Classification of SF298** unclassified

Classification of Abstract unclassified	**Limitation of Abstract** unlimited
Number of Pages 18	

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 20-April-2001	3. REPORT TYPE AND DATES COVERED Final Report

4. TITLE AND SUBTITLE Neural Networks For Robot Control	5. FUNDING NUMBERS F61775-00-WE022
6. AUTHOR(S) Professor Chaiban Nasr	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Lebanese University-Faculty of Engineering, Section I El-Kobbeh Tripoli North Lebanon	8. PERFORMING ORGANIZATION REPORT NUMBER N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) EOARD PSC 802 BOX 14 FPO 09499-0200	10. SPONSORING/MONITORING AGENCY REPORT NUMBER SPC 00-4022
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE A

13. ABSTRACT (Maximum 200 words)

This report results from a contract tasking Lebanese University-Faculty of Engineering, Section I as follows: The proposed research consists of the following: a) Application of artificial neural networks (multi-layer perceptrons, MLPs) for 2D planar robot arm by using the dynamic back-propagation methods for the adjustment of parameters; and optimization of the architecture; b) Application of artificial neural networks in controlling closed-loop 2D planar robot arm and comparison with the use of proportional-integral-differential (PID) controllers. The results should create a basis for a further research program connecting the fundamental knowledge with practical applications that can subsequently include: 1) Theoretical and experimental studies in controlling dynamic robot arms by using neural networks in real-time process; 2) Research of optimal architectures used in closed-loop systems in order to compare with adaptive and robust control.

14. SUBJECT TERMS EOARD, Control, Robotics, Neural Networks	15. NUMBER OF PAGES 18
	16. PRICE CODE N/A

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19, SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

some bounded uncertain disturbances or plant non-linearity [3], making zero tracking accuracy no longer guaranteed since the steady state tracking error can only be shown to stay within an unknown ball whose size depends on the disturbances. Recently, suggestions have been made to use neural networks to overcome the need for linear parameterization models. In many schemes, neural networks are firstly "trained" off-line to obtain sufficiently accurate approximations of the input-output functions of the systems, and the networks can be appropriately used to construct controllers. Other attempts were made to train the neural network on-line during normal system operations [4,5]. It is known that neural networks have the static mapping capability. However, when they are used as controllers, they are able to realize the dynamics [6]. Despite the fact that neural networks are very powerful in learning complicated dynamics, the sizes of the networks are known to be very large, especially the dynamic ones, which subsequently leads to the need of powerful computational facilities. Recent research [7] on parsimonious construction algorithm for linear in the parameters neural networks overcomes the curse of dimensionality associated with dynamic neural networks. As this is a non-linear optimization scheme, it can only be achieved off-line.

The goal of the present work is the design of three types of Multi-layered neural network controllers applied to control a dynamic 2D-Robot manipulator. During this design, different learning techniques will be used for temporal process, and a study of the behavior of the system will be investigated. The proposed architectures for training different neural network controllers are used to provide the appropriate inputs to the 2D-Robot arm so that the desired response can be obtained. The first architecture is a neural network controller that anticipate the output of a PD controller and which weights are tuned due to the Torque control. The second architecture is the same as the first one but in which the weights of the neural network are tuned by minimizing the output of a PD controller. The third architecture will use an artificial neural network controller that anticipates the desired input of the closed loop system consisting of a PD controller in cascade with a 2D-Robot arm. The search of optimum architectures of the controllers will be carried out. The performances of the three controllers with different trajectories will be examined. Finally, a comparative study with respect to desired performances will be investigated.

2 - 2D DYNAMIC PLANAR ARM:

2-1 - Parameters of the planar arm:

Our focus in this section is to provide the design fundamentals for neural networks controllers used in robotics systems. Trajectory control of robotic manipulators traditionally consists of following a preprogrammed sequence of end effector movements. Robot control usually requires control signals applied at the joints of the robot while the desired trajectory, or sequence of arm end positions, is specified for the end effector.

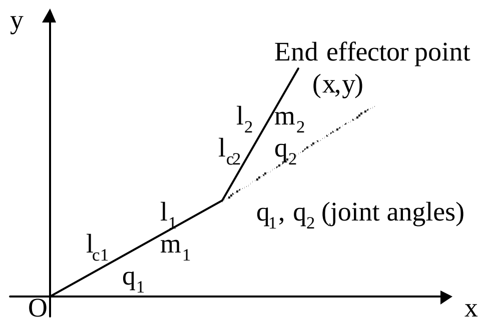

Fig. 1: - Dynamic planar robot arm.

Figure 1 shows a planar manipulator with two revolute joints. Let us fix notation as follows:
For i = 1,2

- q_i denotes the joint angle, which also serves as a generalized coordinate.
- m_i (kg) denotes the mass of link i ($m_1=9.5$; $m_2=5$).
- l_i (m) denotes the length of link i ($l_1=0.25$; $l_2=0.16$).
- l_{ci} (m) denotes the distance from the previous joint to the center of mass of link i ($l_{c1}=0.2$; $l_{c2}=0.14$).
- I_i (kgm^2) denotes the moment of inertia of link i about an axis coming out of the page and passing through the center of mass of link i ($I_1=4.3x10^{-3}$; $I_2=6.1x10^{-3}$).

2-2 - Equation of motion:

In the case of the planar robotic manipulators, two conditions are satisfied:
First, the kinetic energy is a quadratic function of the vector $\dot{\vec{q}}$ of the form:

$$K = \frac{1}{2}\dot{\vec{q}}^T D(\vec{q})\dot{\vec{q}} \qquad (1)$$

Where the n x n "inertia matrix" D is symmetric and positive definite for each $\vec{q}$.

Second, the potential energy V = V ($\vec{q}$) is independent of $\dot{\vec{q}}$.
By applying the Euler-Lagrange equations for such a system, we obtain:

$$\sum_j d_{kj}(\vec{q})\ddot{q}_j + \sum_{i,j} c_{ijk}(\vec{q})\dot{q}_i\dot{q}_j + \phi_k(\vec{q}) = \tau_k, \quad k = 1,\dots,n \qquad (2)$$

In the above equation, there are three types of terms. The first involve the second derivative of the generalized coordinates. The second are quadratic terms in the first derivatives of $\vec{q}$, where the coefficients may depend on $\vec{q}$. These are classified into two types. The Centrifugal terms involve a product of the type $\dot{q}_i^2$, while the Coriolis terms involve a product of the type $\dot{q}_i\dot{q}_j$ where i≠j. The third type of terms are those involving only $\vec{q}$ but not its derivatives. Clearly this third term arise from differentiating the potential energy. It is common to write this equation in matrix form as:

$$D(\vec{q})\ddot{\vec{q}} + C(\vec{q},\dot{\vec{q}})\dot{\vec{q}} + \vec{g}(\vec{q}) = \vec{\tau} \qquad (3)$$

By applying the mechanical equation of motion to the manipulator, we obtain:

$$\vec{u} - B\dot{\vec{q}} - \vec{\tau} = J\ddot{\vec{q}} \qquad (4)$$

Where B is a friction matrix, J is a diagonal inertia matrix and $\vec{u}$ is the applied input torque to the planar manipulator.

Combining equations (3) and (4) yields:

$$M(\vec{q})\ddot{\vec{q}} + \vec{h}(\vec{q},\dot{\vec{q}}) + \vec{g}(\vec{q}) = \vec{u} \qquad (5)$$

Where:

* $\vec{q}, \dot{\vec{q}}, \ddot{\vec{q}}$ are the joint angle, joint velocity, and joint acceleration respectively.
* $M(\vec{q})$ defines a 2x2 mass matrix that describes the inertial properties of the arm.
* $\vec{h}(\vec{q},\dot{\vec{q}}) = (C(\vec{q},\dot{\vec{q}}) + B)\dot{\vec{q}}$ defines a vector that describes Coriolis and Centripetal acceleration, and friction terms in the dynamics of the robot manipulator.
* $\vec{g}(\vec{q})$ is a vector that specifies the effects of gravitational forces acting on the arm.
* $\vec{u}$ represents the input vector torque that act on the manipulator.

<u>**2-3 - Robot dynamic modeling:**</u>

We will make effective use of the Jacobian expressions in computing the kinematic energy. Since we are using joint variables as the generalized coordinates, it follows that:

The expression of the velocity of the center of mass of link one is:

$$\vec{v}_{c1} = J_{\vec{v}_{c1}}\dot{\vec{q}} \qquad (6)$$

Where $J_{\vec{v}_{c1}}$ represent the Jacobian of link one whose expression is give by:

$$J_{\vec{v}_{c1}} = \begin{bmatrix} -\ell_{c1}\sin q_1 & 0 \\ \ell_{c1}\cos q_1 & 0 \\ 0 & 0 \end{bmatrix} \qquad (7)$$

Similarly, the expression of the velocity of the center of mass of link two is:

$$\vec{v}_{c2} = J_{\vec{v}_{c2}}\dot{\vec{q}} \qquad (8)$$

Where $J_{\vec{v}_{c2}}$ represent the Jacobian of link two whose expression is give by:

$$J_{\vec{v}_{c2}} = \begin{bmatrix} -\ell_1\sin q_1 - \ell_{c2}\sin(q_1+q_2) & -\ell_{c2}\sin(q_1+q_2) \\ \ell_1\cos q_1 + \ell_{c2}\cos(q_1+q_2) & \ell_{c2}\cos(q_1+q_2) \\ 0 & 0 \end{bmatrix} \qquad (9)$$

Hence the translational part of the kinetic energy is:

$$\frac{1}{2}m_1\vec{v}_{c1}^T\vec{v}_{c1}+\frac{1}{2}m_2\vec{v}_{c2}^T\vec{v}_{c2}=\frac{1}{2}\dot{\vec{q}}^T\left\{m_1 J_{\vec{v}_{c1}}^T J_{\vec{v}_{c1}}+m_2 J_{\vec{v}_{c2}}^T J_{\vec{v}_{c2}}\right\}\dot{\vec{q}} \quad (10)$$

Next we deal with the angular velocity terms. Because of the particularly simple nature of this manipulator, many of the potential difficulties do not arise.
First, it is clear that:

$$\vec{\omega}_1=\dot{q}_1\vec{k},\ \ \vec{\omega}_2=(\dot{q}_1+\dot{q}_2)\vec{k} \quad (11)$$

When expressed in the base inertial frame.

The z-axes of all of these frames are in the same direction, so the above expression is also valid in the link-bound frame. Moreover, since $\vec{\omega}_i$ is aligned with $\vec{k}$, the triple product $\vec{\omega}_i^T I_i \vec{\omega}_i$ reduces simply to $(I_{33})_i$ times the square of the magnitude of the angular velocity. This quantity $(I_{33})_i$ is what we have labeled I_i above. Hence the rotational kinetic energy of the overall system is:

$$\frac{1}{2}\dot{\vec{q}}^T\left\{I_1\begin{bmatrix}1&0\\0&0\end{bmatrix}+I_2\begin{bmatrix}1&1\\1&1\end{bmatrix}\right\}\dot{\vec{q}} \quad (12)$$

Finally, the expressions of the different elements of the inertia matrix D are:

$$\begin{aligned}d_{11}&=m_1\ell_{c1}^2+m_2(\ell_1^2+\ell_{c2}^2+2\ell_1\ell_{c2}\cos q_2)+I_1+I_2\\ d_{12}&=d_{21}=m_2(\ell_{c2}^2+\ell_1\ell_{c2}\cos q_2)+I_2\\ d_{22}&=m_2\ell_{c2}^2+I_2\end{aligned} \quad (13)$$

The Christoffel symbols are defined by the expressions:

$$\begin{aligned}c_{111}&=\frac{1}{2}\frac{\partial d_{11}}{\partial q_1}=0\\ c_{121}&=c_{211}=-m_2\ell_1\ell_{c2}\sin q_2\\ c_{221}&=-m_2\ell_1\ell_{c2}\sin q_2\\ c_{112}&=m_2\ell_1\ell_{c2}\sin q_2\\ c_{122}&=c_{212}=c_{222}=0\end{aligned} \quad (14)$$

The potential energy of the manipulator is just the sum of those of the two links. For each link, the potential energy is just its mass multiplied by the gravitational acceleration and the height of its center of mass. Hence, the functions ϕ_k become:

$$\begin{aligned}\phi_1&=(m_1\ell_{c1}+m_2\ell_1)g\cos q_1+m_2\ell_{c2}g\cos(q_1+q_2)\\ \phi_2&=m_2\ell_{c2}g\cos(q_1+q_2)\end{aligned} \quad (15)$$

Finally, the dynamical equations of the system are:

$$
\begin{aligned}
&d_{11}\ddot{q}_1 + d_{12}\ddot{q}_2 + c_{121}\dot{q}_1\dot{q}_2 + c_{211}\dot{q}_2\dot{q}_1 + c_{221}\dot{q}_2^2 + \phi_1 = \tau_1 \\
&d_{21}\ddot{q}_1 + d_{22}\ddot{q}_2 + c_{112}\dot{q}_1^2 + \phi_2 = \tau_2 \qquad (16)
\end{aligned}
$$

The robot model is simulated numerically using the fourth-order Runge-Kutta method for the solution of differential equations. Note that the robot dynamics equation (16) must first be rearranged into standard form:

$$\ddot{\vec{q}} = M^{-1}(\vec{q})(\vec{u} - h(\vec{q},\dot{\vec{q}}) - \vec{g}(\vec{q})) \qquad (17)$$

Implementation of the numerical algorithm is then straightforward.

3 - Multi-Layer Perceptrons Controllers:

3 -1- Multi-layer perceptrons (MLP):

Multi-layer perceptrons [8] provide one arrangement for neural network implementation, by means of nonlinear relationships between, firstly, the network inputs to outputs and, secondly, the network parameters to outputs. Such a network consists of a number of neuron layers, n, linking its input vector, u, to its output vector, y, by means of the equation:

$$y = \varphi_n(W_n\varphi_{n-1}(W_{n-1}\ldots\varphi_1(W_1u + b_1) +\ldots+ b_{n-1}) + b_n) \qquad (18)$$

In which W_i is the weight matrix associated with the layer, φ_i is a nonlinear operator associated with the ith layer and b_i indicate threshold or bias values associated with each node in the ith layer. The function φ_i is a sigmoid function for all n.

It is known in reality that real neurons, located in different areas of the nervous system, have different modes of behavior [9] ranging from Gaussian-like for visual needs to sigmoid for ocular motor needs. It is generally the case for artificial neural networks, however, that only one type of non-linearity is employed for a particular network, this linking in closely with the fact that each network is only employed for one particular task.

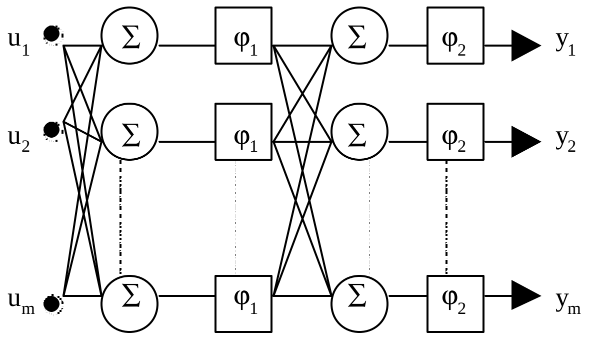

Fig.2: - A two-layer multi-layer perceptron.

Figure 2 shows a fully connected network in that all of the neuron outputs in one layer of the network are connected as inputs to the next layer. This is normal practice: however, it is quite possible for part-connectivity to be realized by connecting a group of outputs to only specific input. By this means sub-models can be formed within the overall MLP, and these can be particularly useful where a specific system characteristic is to be dealt with or inputs/outputs from the system can be categorized into certain types.

Whatever the network connectivity, key questions in the use of MLPs are how many layers there should be and how many neurons there should be in each layer. Once these structural features of the network have been selected, it remains for the adjustable weights to be settled on such that the network is completely specified in terms of its functionality.

<u>3-2 - Dynamic back propagation learning:</u>

By far the most popular method employed for weight training in MLP neural networks is called back propagation [10]. In the standard feed-forward MLP network, back propagation solves the problem of missing information to the hidden layers, neither the input to nor the reference signals for the hidden layers are known. This solving problem is obtained by taking the inputs to the hidden layers as being the inputs to the first layers propagated through the network. The reference signals for the hidden layers are then obtained by error back propagation through the network. This is realized by obtaining the partial derivative of the squared error with respect to the parameters.

It is worth pointing out that the back propagation algorithm has also been used for weight learning in feedback neural networks [11,12], these being networks in which the network structure incorporates feedback, whereby the output of every neuron is fed back, in weighted form, to the input of every neuron. The architecture of such a network is inherently dynamic and realizes powerful capabilities due to its complexity.

Consider, as a starting point, a single neuron with output y_i; then

$$y_i = \varphi(x_i) = \frac{1-\exp(-x_i)}{1+\exp(-x_i)} \tag{19}$$

In which:

$$x_i = \sum_{i=1}^{m} w_{ij}u_j + w_0 \tag{20}$$

In this expression w_0 is a bias term. If it is assumed that at an instant in time, for an input u_j the output y_i should be equal to the desired output y_d, then the squared error of the output signal is given by:

$$E_i = \frac{1}{2}(y_d - y_i)^2 = \frac{1}{2}e_i^2 \tag{21}$$

and it is desired to minimize E_i by means of a suitable/best choice of the weighting coefficients w_{ij} .

Consider the problem of minimizing the scalar error function E(W), where W is a vector of weights to be adjusted by means of an interactive procedure generating a number of search points, W(k), such that:

$$W(k+1) = \alpha(k)W(k) + \eta(k)d(k) \tag{22}$$

In this relation an initial set of weightings W(0) is made through prior knowledge, by a reasoned guess or even relatively randomly. The term $\alpha(k)W(k)$ represent a momentum term and $\alpha(k)$ is usually a positive number called the momentum constant. The term d(k) indicates the search direction, whereas $\eta(k)$ indicates the length of search step or the amount of learning to be carried out.

In this way, the weights associated with one neuron can be adjusted in order to minimize the squared error. The approach can then be extended in order to adjust all of the weights in the MLP network. So, overall, a set of input/desired output data values are used to train the entire network. The input set also realizing a corresponding set of network weights such that the error between the desired output signals and the actual network output signals is minimized in terms of the average overall learning points. The back propagation algorithm employs the steepest-descent method to arrive at a minimum of the mean squared error function. For one specific data pair, the error squared can be written as:

$$E_k = \frac{1}{2}\sum_{i=1}^{m}(y_{dk} - y_{ik})^2 = \frac{1}{2}\sum_{i=1}^{m} e_{ik}^2 \tag{23}$$

where m neurons are assumed to be present, and y_{ik} is the ith neuron's kth output value.

The global error is then found by minimizing E_k over all the data set. If the number of data values that are present is N, then we have:

$$E = \sum_{k=1}^{N} E_k \tag{24}$$

This error function can then be minimized in batch mode or recursively in an on-line manner.

The network is now fully trained on the data presented and can be employed with any further data, although it may be desirable to present the data again cyclically until the overall error falls below a previously defined minimum value, i.e. until the weights converge. An important feature then emerges in that the MLP network has the ability to generalize when it is presented with new data not previously dealt with.

<u>3-3 - NN controller tuned by the torque control (First Method):</u>

Before describing the neural network systems, we detail the training procedure used to provide error information to the systems.

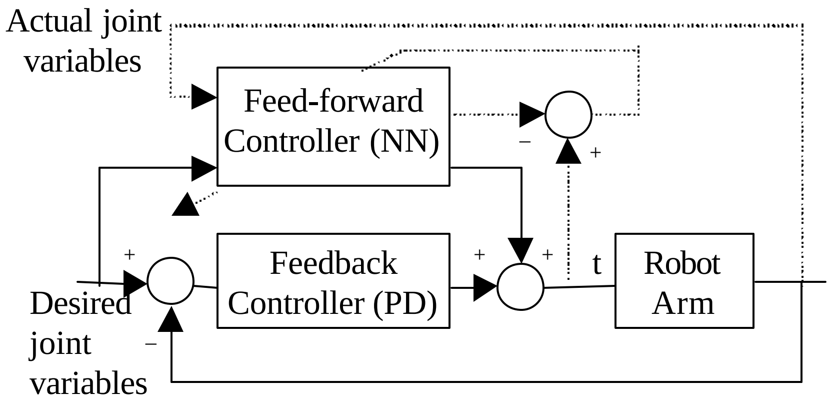

Fig. 3: - Feed-forward controller trained by minimizing the torque error

The procedure involves training an adaptive feed-forward controller to control the arm in conjunction with a fixed feedback controller (Figure 3). The feedback controller aids in generating training data that the forward controller uses to learn a model of the arm's inverse dynamics. This direct approach to training a neural network controller has been studied [13]. In order to achieve a desired acceleration $\vec{\ddot{q}}$, an appropriate torque $\vec{\tau}(t)$ must be applied to the arm. The relationship between acceleration and torque is the inverse dynamics of the arm and the goal of the learning procedure is to train a feed-forward controller to model this relationship. The feed-forward controller is trained on-line in the following manner. At each time step, the control signal is obtained by summing the outputs of the feed-forward controller and the feedback controller. The inputs to the feed-forward controller are the desired joint positions, velocities, and accelerations for the current time step, as specified by the reference trajectory. The inputs to the feedback controller (a PD controller) are the desired and actual joint positions and velocities. The sum of the feed-forward and feedback control signals is a torque vector that is applied to the arm (figure 3, straight line). The resulting joint accelerations are observed and the feed-forward controller then receives the actual joint positions, velocities, and accelerations as inputs and computes new outputs (figure 3, dashed line). An error is computed between this output and the actual torque applied to the arm, and the error is used to change the weights in the controller using back-propagation algorithm [14]. Early in the training session, the feedback controller dominates and the arm follows the desired trajectory imprecisely. As the feed-forward controller learns to model the arm's inverse dynamics, it begins to generate torque that allows the arm to follow the desired trajectory more faithfully.

The neural network used is a multi-layer perceptron with one hidden layer. The input vector contains nine components that are real valued and represent the robot arm's joint positions, velocities, and accelerations.

$$\begin{gathered}\ddot{q}_1,\ \ddot{q}_2,\ \ddot{q}_1\cos(q_2),\ \ddot{q}_2\cos(q_2),\ \dot{q}_1^2\sin(q_2)\\ \dot{q}_2^2\sin(q_2),\ \dot{q}_1\dot{q}_2\sin(q_2),\ \cos(q_1),\ \cos(q_1+q_2)\end{gathered} \qquad (25)$$

Those components (25) are the transformations of the joint variables that are conveniently chosen so as to facilitate the learning process of the robot arm's inverse dynamics [14]. The output layer has two neurons, and the number of hidden neurons in the hidden layer is chosen by simulation. The activation function used is a sigmoid that uses the hyperbolic tangent function.

3-4- NN controller tuned by minimizing a torque control (Second Method):

This second method [13,15] is represented in figure 4.

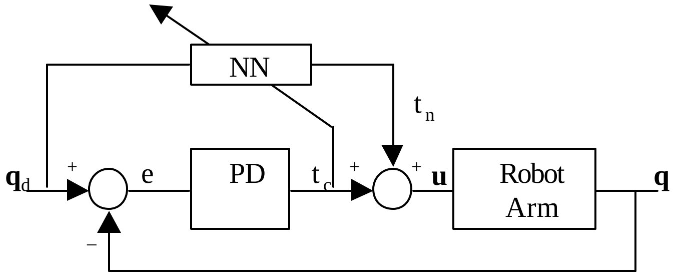

Fig. 4: - Feedback Error Learning Control Structure.

The technique presented here employs a standard feed-forward neural network. The neural network output is $\vec{\tau}_n$ and the control law of the arm is given by the expression:

$$\vec{u} = \vec{\tau}_c + \vec{\tau}_n \qquad (26)$$

Where: $\vec{\tau}_c = k_v \dot{\vec{\varepsilon}} + k_p \vec{\varepsilon}$ and $\vec{\varepsilon} = \vec{q}_d - \vec{q}$ (27)

Combining the robot dynamic equation (5) and the control law (26), the error dynamic equation can be written as:

$$k_v \dot{\vec{\varepsilon}} + k_p \vec{\varepsilon} + \vec{\tau}_n = M\ddot{\vec{q}} + \vec{h}(\vec{q}, \dot{\vec{q}}) + \vec{g}(\vec{q}) \qquad (28)$$

The training signal of the neural network is the output $\vec{\tau}_c$ of the PD controller. From equation (28) the expression of the output $\vec{\tau}_c$ will be:

$$\vec{\tau}_c = M\ddot{\vec{q}} + \vec{h}(\vec{q}, \dot{\vec{q}}) + \vec{g}(\vec{q}) - \vec{\tau}_n \qquad (29)$$

The neural network is trained to minimize $E = \frac{1}{2}\vec{\tau}_c^T \vec{\tau}_c$. Thus, at convergence ($\vec{\tau}_c = \vec{0}$), the ideal output of the neural network will represents the robot inverse dynamics. The neural network used is a multi-layer perceptron with one hidden layer. The input vector contains four components $q_1, q_2, \dot{q}_1, \dot{q}_2$ that are real valued and represent the robot arm's joint positions, and velocities. The output layer has two neurons, and the number of hidden neurons in the hidden layer is chosen by simulation. The activation function used is a sigmoid that uses the hyperbolic tangent function.

3-5 - NN controller that anticipates the desired input (Third Method):

The third architecture uses an artificial neural network controller that anticipates the desired input of the closed loop system consisting of a PD controller in cascade with the robot arm as shown in figure 5.

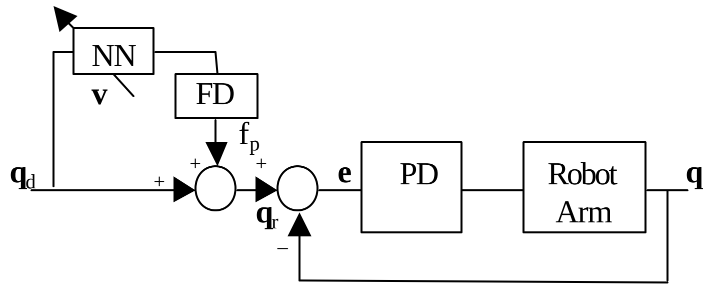

Fig. 5: - Neural Network that anticipates the desired input.

In this structure the neural network controller is used to modify the desired trajectory (q_{d1}, q_{d2}) instead of generating a compensating torque as in the two previous methods. It has been shown [16] that the trajectory modification approach for compensating robot model uncertainties in Cartesian space motion control is as effective as compensating the control torque using neural network. The purpose here is to develop the same control scheme for the non-model based PD control problem. Let ϕ_p be the neural network output. The closed loop system equation is given by:

$$k_v\dot{\vec{e}} + k_p\vec{e} = M\ddot{\vec{q}} + \vec{h}(\vec{q},\dot{\vec{q}}) + \vec{g}(\vec{q}) \qquad (30)$$

where $$\vec{e} = \vec{q}_r - \vec{q} = \vec{q}_d + \vec{\phi}_p - \vec{q} = \vec{\varepsilon} + \vec{\phi}_p \qquad (31)$$

and $$\dot{\vec{e}} = \dot{\vec{q}}_r - \dot{\vec{q}} = \dot{\vec{q}}_d + \dot{\vec{\phi}}_p - \dot{\vec{q}} = \dot{\vec{\varepsilon}} + \dot{\vec{\phi}}_p \qquad (32)$$

The expression $\dot{\vec{\phi}}_p$ can be estimated from the finite difference:

$$\dot{\vec{\phi}}_p = (\vec{\phi}_p(t) - \vec{\phi}_p(t-1))/T \qquad (33)$$

T is the sampling period.

Substituting the error and its derivative into equation (30) yields:

$$k_v\dot{\vec{\varepsilon}} + k_p\vec{\varepsilon} = M\ddot{\vec{q}} + \vec{h}(\vec{q},\dot{\vec{q}}) + \vec{g}(\vec{q}) - k_v\dot{\vec{\phi}}_p - k_p\vec{\phi}_p \qquad (34)$$

We choose the training signal $\vec{v}$ as $\vec{v} = k_v\dot{\vec{\varepsilon}} + k_p\vec{\varepsilon}$. The neural network output is trained to satisfy $\vec{v} = \vec{0}$. Thus the neural network output combined with its derivative will represent the inverse dynamics of the robot. The attractive feature of this scheme is that it can be easily implemented in the trajectory planner that is outside the closed loop system. The neural network is trained to minimize $E = \frac{1}{2}\vec{v}^T\vec{v}$. The neural network used is a multi-layer perceptron with one hidden layer. The input vector contains four components $q_1, q_2, \dot{q}_1, \dot{q}_2$ that are real valued and represent the robot arm's joint positions, and velocities. The output layer has two neurons, and the number of hidden neurons in the hidden layer is chosen by simulation. The activation function used is a sigmoid that uses the hyperbolic tangent function.

4 - NUMERICAL SIMULATION AND COMPARISON:

First, we simulate the robot arm without using neural network controller. We have considered a desired trajectory defined by the equations:

$$Y_m = 0.2 + 0.075\sin(2\pi t/3) \text{ and } Z_m = 0.2 + 0.075\cos(2\pi t/3) \qquad (35)$$

We have used $\Delta t = 0.01$seconds for time integration. The choice of the 2x2 gain matrices used in the PD controller was to obtain a stable system in the closed loop system with bad performances, $k_v = 10.I$ and $k_p = 100.I$ where I represents the identity matrix of order two. Figure 8 (blue line) shows the simulated response of the system for the desired trajectory. Figures 9 & 10 (blue line) show the responses of the joints angle and joints velocity. It is clear that the choices of those values of gains are not well adapted to the control of this dynamic 2-D robot arm.

4-1 - Numerical simulation of the First Method:

During the numerical simulation of the three methods, we have used the back-propagation method with the first order convergence algorithm for training the neural network in order to determine the synaptic weights. The objective was to minimize a cumulative error E defined by:

$$E = \sum_{n=1}^{N}[(y_d(n\Delta t) - y(n\Delta t))^2 + (z_d(n\Delta t) - z(n\Delta t))^2] \qquad (36)$$

where $N = t/\Delta t$ denotes the sampling number within one trial. $y_d(n\Delta t), z_d(n\Delta t)$ are the demands of the trajectory on the Y-Z plane at the sampling time $n\Delta t$, and $y(n\Delta t), z(n\Delta t)$ are the actual trajectories.

In the first method, we have trained a number of neural networks using the error back-propagation technique with different learning rates and different momentum terms. Figure 7 shows an optimum learning rate equal to 3.10^{-3} and an optimum momentum term equal to 0.1. Figure 6 shows that the (9-18-2) multi-layer perceptron represents the optimal architecture with minimum cumulative error equal to $4.27\text{x}10^{-5}$ obtained after 40 cycles. Moreover, figure 8 (green line) shows the tracking response of the end effector to the desired trajectory, and figure 11 shows the evolution of the instantaneous square error between the desired and actual position. This method presents the best convergence in comparison with the other methods.

4-2 – Numerical simulation of the Second Method:

A number of neural network controllers have been trained using the error back-propagation technique with different learning rates and different momentum terms. Figure 7 shows an optimum learning rate equal to 10^{-3} and an optimum momentum term equal to 0.1. Figure 6 shows that the (4-18-2) multi-layer perceptron represents the optimal architecture with minimum cumulative error equal to 1.13×10^{-4} obtained after 40 cycles. Moreover, figure 10 (green line) shows the responses of the joints velocities to the desired inputs, and figure 11 shows the evolution of the instantaneous square error between the desired and actual position.

4-3 – Numerical simulation of the Third Method:

This new scheme is different from the two other approaches in that it modifies the desired trajectory before inputting to the robot PD controller. A number of neural network controllers have been trained using the error back-propagation technique with different learning rates and different momentum terms. Figure 7 shows an optimum learning rate equal to 10^{-6} and an optimum momentum term equal to 0.1. Figure 6 shows that the (4-6-2) multi-layer perceptron represents the optimal architecture with minimum error equal to 1.3×10^{-3} obtained after 40 cycles. Moreover, figure 9 (green line) shows the responses of the joints positions to the desired inputs, and figure 11 shows the evolution of the instantaneous square error between the desired and actual position.

4-4 – Comparison between the three methods:

By making a comparison between the three methods with respect to the cumulative error, figure 6 shows that the first method is the best in performances. Its cumulative error is 2.5 times less than the second method and 5 times less than the third method. But the third method needs, in its optimal architecture, less parameter than the two others do. While the first method needs 218 parameters and the second method 128 parameters, the third method need only 44 parameters. Figure 11 shows the instantaneous temporal square error obtained by the three methods between the desired trajectory and the actual response. It is clear that the first method is the best in accuracy.

5 - COMMENTS AND PERSPECTIVES:

Applications in new technologies such as robotics, manufacturing, space technology, and medical instrumentation, as well as those in older technologies such as process control and aircraft control, are creating a wide spectrum of control problems in which non-linearity, uncertainties, and complexity play a major role [17]. For the solution of many of these problems techniques based on artificial neural network are beginning to complement conventional control techniques [18], and in some cases they are emerging as the only viable alternatives. From a control theoretic point of view, artificial neural network may be considered as tractable parameterized families of nonlinear maps. As such they have found wide application in pattern recognition problems, which require nonlinear decision surfaces. With the introduction of dynamics and feedback, the scope of such networks as identifiers and controllers in nonlinear dynamical systems has increased significantly.

Our works aims to study three types of neural network controllers for trajectory control of robotic manipulators, and to investigate the performances of the application of these controllers (multi-layer perceptrons, MLP) for closed loop 2D planar robot arm by using the back-propagation methods for the adjustment of parameters. We have implemented the different algorithms and we have obtained the optimal architecture in the three different methods. Good results where obtained through the convergence of the algorithms. The first method presents the smallest cumulative error in comparison with the two other methods (figure 11) and the best response-time for the joint angle and velocity. But the third method uses less input components and hidden neurons making the real-time process faster. Moreover, this third method presents practical advantages over the two other methods in that compensation is done outside the control loop so it can be implemented easily at the command trajectory planning level external to an existing robot controller.

Our target from this work is the real-time process (implementation on DSP of different architectures in neural networks and studying the behavior of all the process).

6 - REFERENCES:

[1]: T. J. Tarn, A. K. Bejczy, A. Isidori, and Y. Chen, "Nonlinear feedback in robot arm control," in Proc. IEEE Conf. Decision and Control. Las Vegas, NV, vol.2, pp.736-751, Dec 1984

[2]: L. L. Whitcomb, A. A. Rizzi, and D. E. Koditschek, "Comparative experiments with a new adaptive controller for robot arm," IEEE Trans. Robotics and Automation, vol. 9, no.1, pp. 59-69, 1993.

[3]: J. S Reed and P. A. Ioannou, "Instability analysis and robust adaptive control of robotic manipulators," IEEE Trans. Robotics and Automation, vol.5, no.3, pp. 381-386, 1989.

[4]: S. S. Ge and T. H. Lee, "Parallel adaptive neural network control of robots," Proc. Inst. Mechanical Engineers, Part I: J. Systems and Control Engineering, vol. 208, pp. 231-237, 1994

[5]: F. L. Lewis, K. Liu, and A. Yesildirek, " Neural net robot controller with guaranteed tracking performance," IEEE Trans. Neural Networks, vol. 6, no.3, pp. 703-715, 1995

[6]: A. U. Levin and K. S. Narendra, “Control of nonlinear dynamical systems using neural networks part II”, IEEE Trans on neural networks, vol.7, pp.31-42, n^{0}1, January 1996.

[7]: S. S. Ge, C. C. Hang, and L. C. Woon, "Adaptive neural network control of robot manipulators in task space," IEEE Trans. Industrial Electronics, vol.44, no.6, pp.746-752, 1997

[8]: K. S. Narendra and K. Parthasarathy, Identification and control of dynamical systems using neural networks, IEEE Trans. on Neural Networks, **1**, pp.4-27 (1990).

[9]: D. H. Ballar, Cortical connections and parallel processing: structure and function. In Vision, brain and cooperative computation, M.Arbib and Hamson (eds.), pp.563-621, MIT Press, Cambridge, MA (1988).
[10]: D. E. Rumelhart and J. L. McClelland (eds.), Parallel distributed processing: explorations in microstructure of cognition, **1**: Foundations, MIT Press, Cambridge, MA (1986).
[11]: F. J. Pineda, Recurrent back propagation and dynamical approach to adaptive neural computation, Neural computation, **1**, pp.162-172 (1989).
[12]: K. S. Narendra and K. Parthasarathy, Gradient methods for the optimization of dynamical systems containing neural networks, IEEE Trans. on Neural Networks, **2**, pp.252-262 (1991).
[13]: M. Kawato, K. Furukawa, and R. Suzuki, "Hierarchical neural-network model for control and learning of voluntary movement," Biological Cybernetics, vol. 57, pp.169-185, 1987
[14]: R. A. Jacobs, and M. I. Jordan, "Learning piecewise control strategies in a modular neural network architecture," IEEE Trans. on Systems, Man, and Cybernetics, vol. 23, no.2, pp.337-345, 1993
[15]: S. Jung, and T. C. Hsia, "New neural network control technique for non-model based robot manipulator control," IEEE Trans. on Systems, Man, and Cybernetics, vol.1, pp.2928-2933, 1995
[16]: S. Jung, and T. C. Hsia, "On reference trajectory modification approach for Cartesian space neural network control of robot manipulators," Proc. of IEEE International Conference on Robotics and Automation, Nagoya, 1995
[17]: Q. Song, J. Xiao, and Y. C. Soh, "Robust Back-propagation Training Algorithm for Multi-layered Neural Tracking Controller," IEEE Trans. on Neural Networks, vol. 10, no.5, pp.1133-1141, 1999.
[18]: K. S. Narendra, Neural Networks for Control: Theory and Practice, Proceedings of the IEEE, vol.84 n°10, (1996).

****

This work was funded by the U. S. Air Force Office of Scientific Research (AFOSR), the European Office of Aerospace Research and Development (EOARD) under the supervision of Dr. Charbel N Raffoul.

****

RESULTS OF THE SIMULATIONS

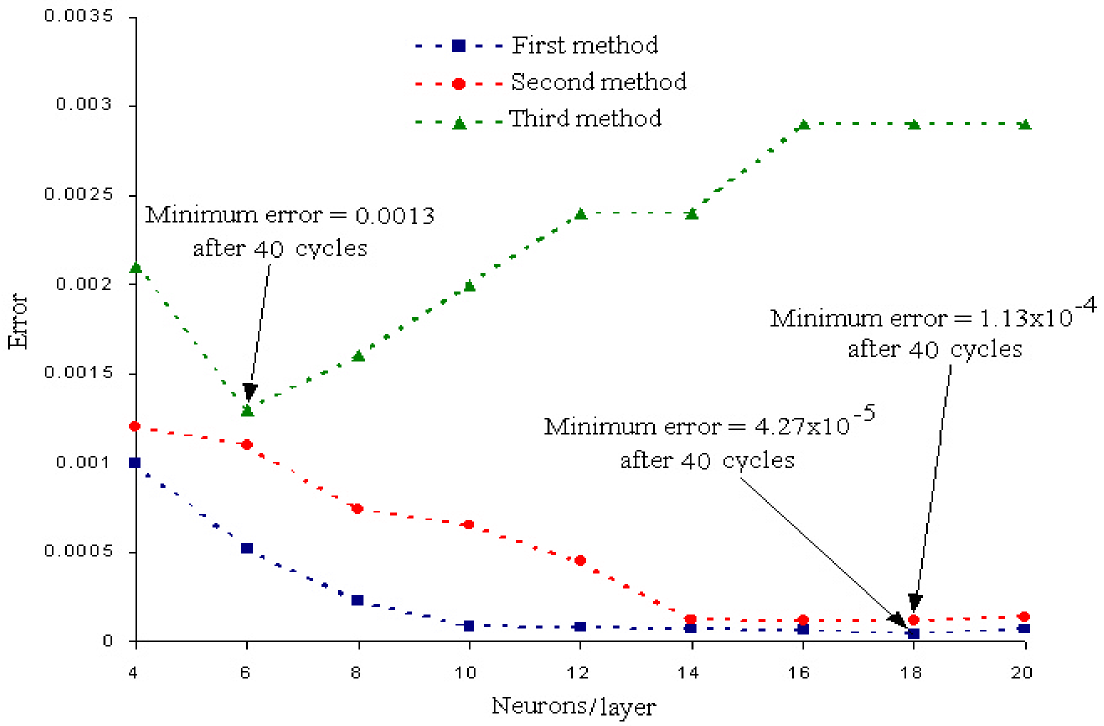

Fig.6: -Optimized architecture of the neural network controller for the three methods

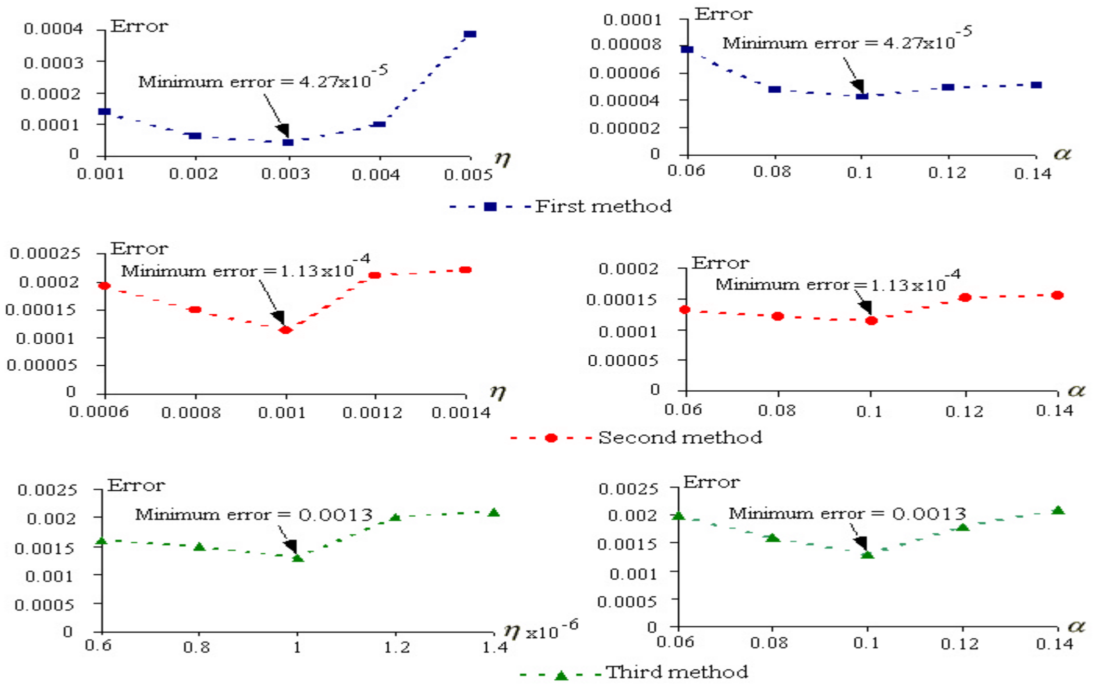

Fig.7: -Optimized learning rate and Momentum term for the three methods

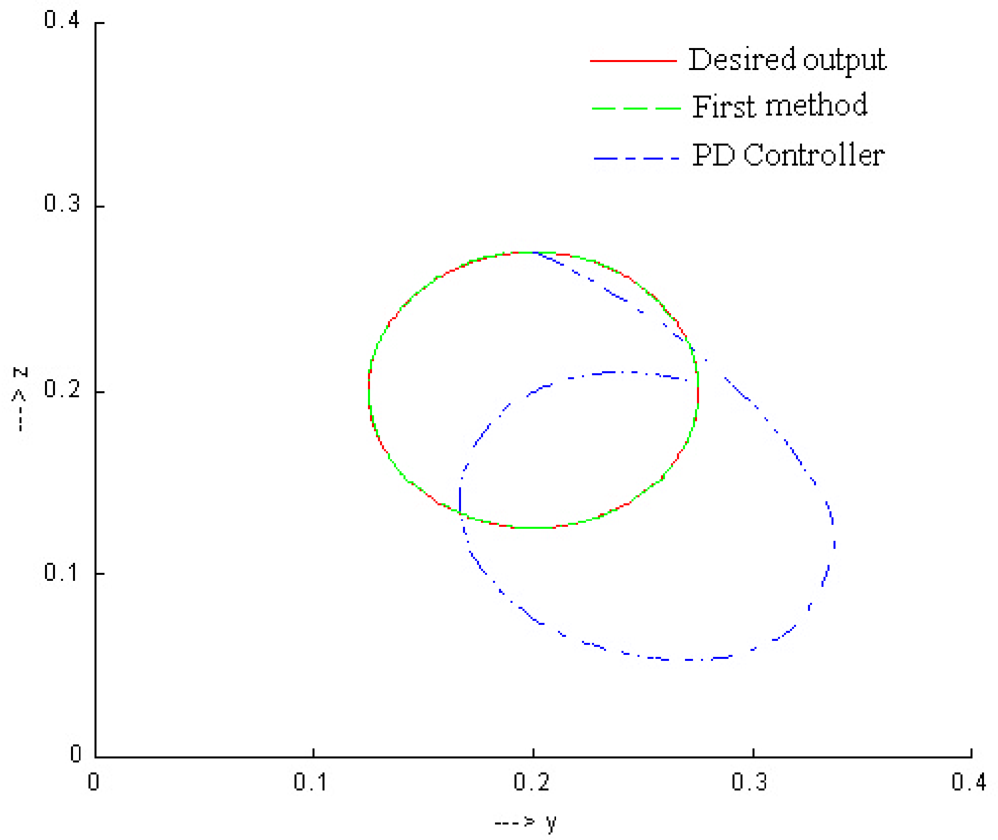

Fig.8: -Simulated trajectory of the end-effector in the yz Cartesian plane

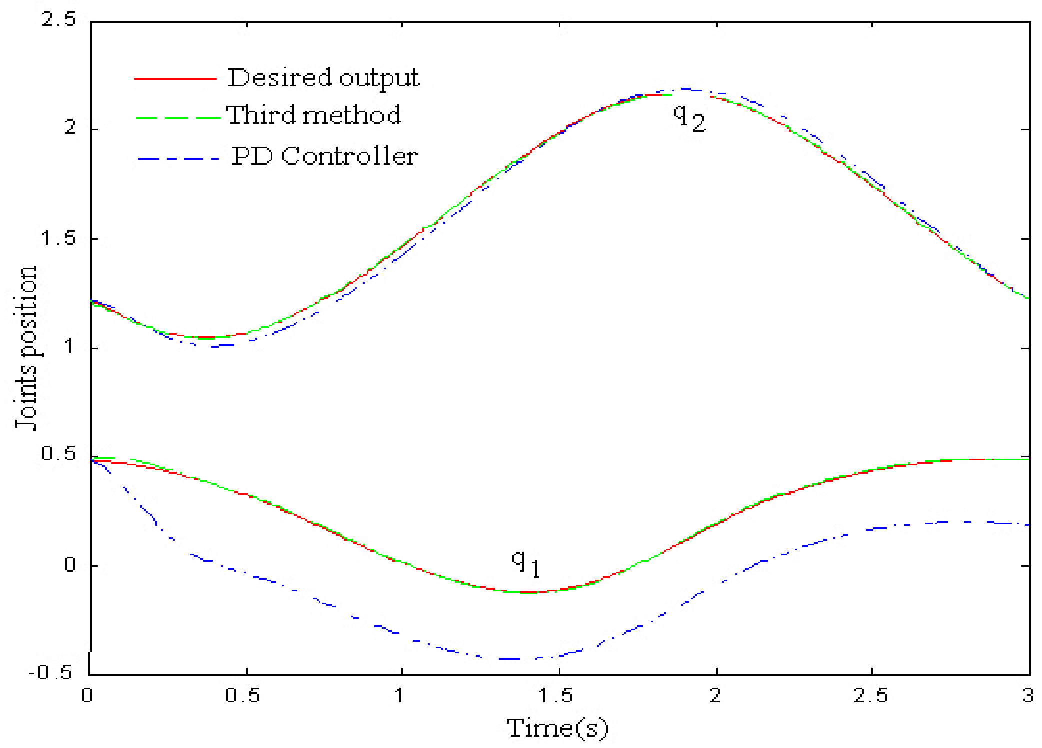

Fig.9: -Desired and actual joints position responses

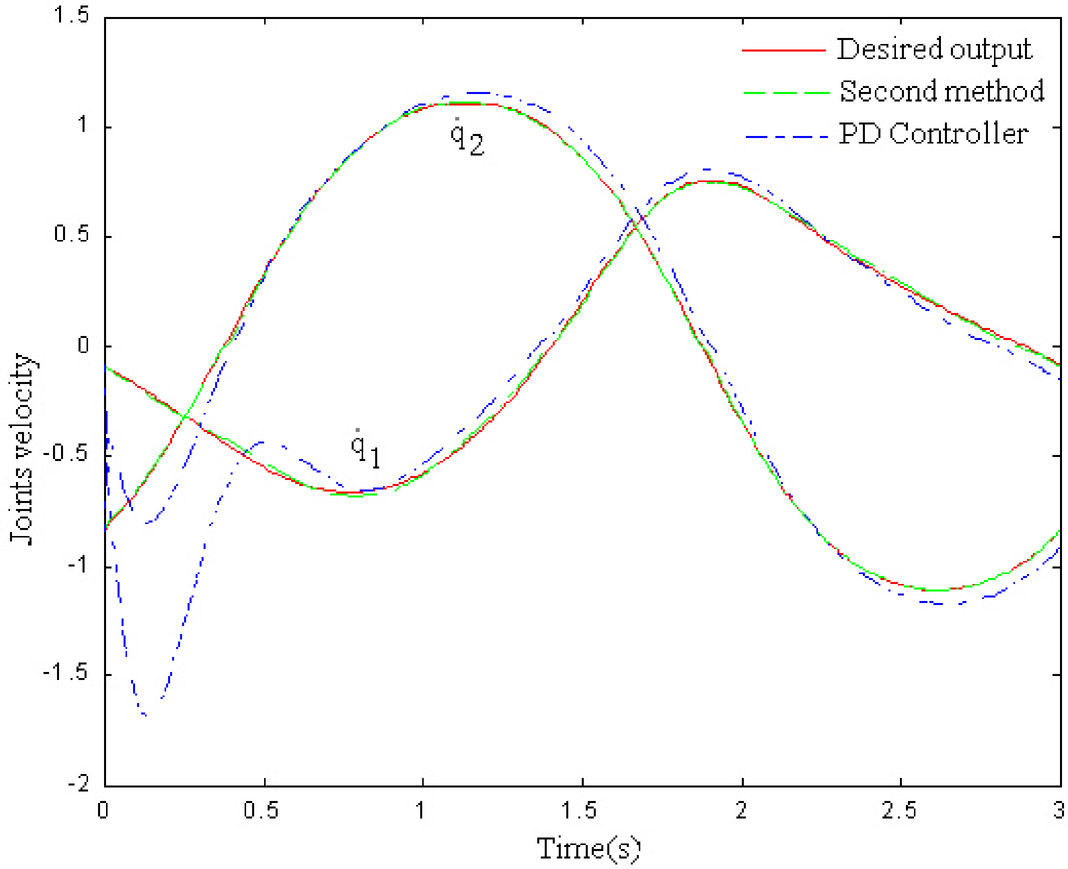

Fig.10: -Desired and actual joints velocity responses

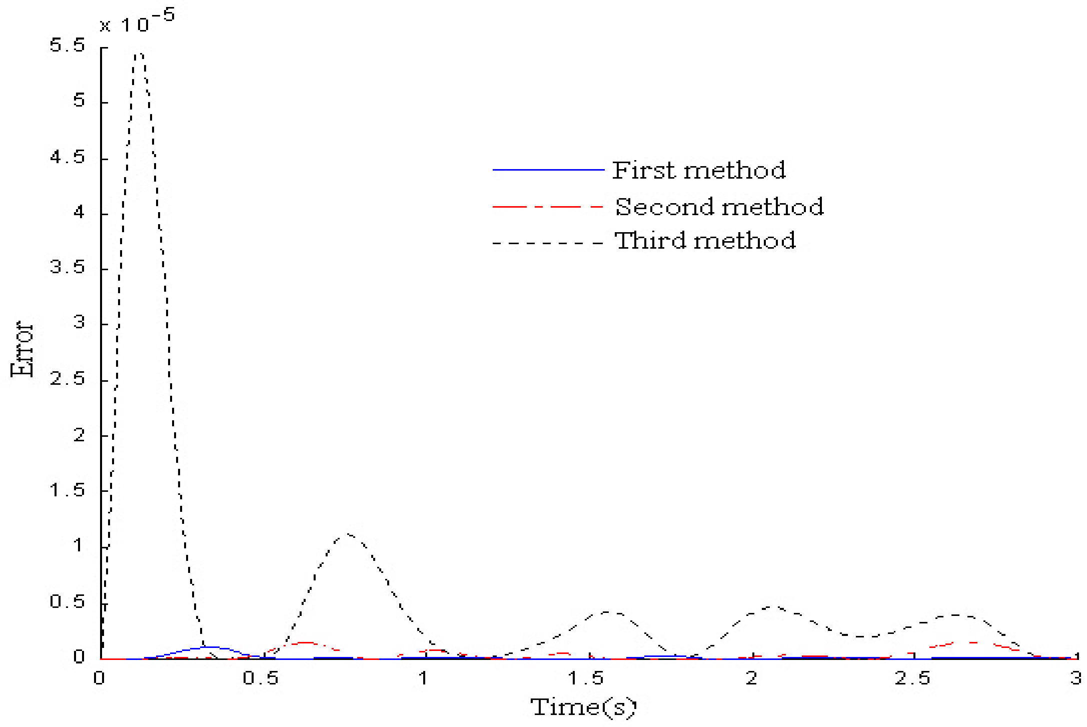

Fig.11: - Instantaneous square error between actual and desired position